LIBERTAS MATHEMATICA, VOL 12 (1992)

TUNING OF THE MEMORY ALLOCATION IN A CLIENT/SERVER NETWORK ARCHITECTURE. GUIDELINES AND RESULTS

Vlorel Vlad

GOALS: The scope of this paper is to present some practical aspects and guidelines for an Oracle RDBMS application tuning. Almost all of them have been applied and verified on a production application in a client/server environment.
Tuning an Oracle system implies different approaches such as: (a) Tune the SQL statements inside the application, (b) Tune the Memory Allocation, (c) Tune the I/O, and (d) Tune the Contention. This paper only treats the second approach on tuning the Memory Allocation.

STATUS: A software application has been installed and running satisfactorily for some time after the system implementation. As transaction volume increases, and tables grow to thousands of rows, the response time of the system has decreased dramatically. A series of "complaints" of screens running "slow" are voiced from the production field. This is a sign that a process of re-evaluation and tuning of the system is needed.

The process of tuning any application software implies different steps, and can be approached from different points of view. The decision of what type of tuning your system needs, is taken after some specific questions are answered, such as:

- Does your system need performance improvement on a whole scale or only some specific modules?
- Are there only some transactions that perform *slow*, or are all of them performing *poor*?
- What is more important: the speed of execution or some cost constraints ? (no budget allocated for eventual memory increase).

The way the user answers these questions can have a big impact on the performance analysis of the system.
This paper will continue some of the steps and tasks defined in "Tuning the SQL Statements in a Oracle client/server application", (see reference number 4), suggested to improve and analyze the performance. The most important factor to start in tuning an application is the Analysis of the SQL Statements. The way you write the *WHERE* and *FROM* clauses makes a big difference in an Oracle environment. The Oracle "Optimizer" will parse and interprete these statements in a specific way, and the degree of understanding of these *expert systems* rules make the application run smooth or not. Tuning the SQL statements according to "Optimizer" rules by using **Trace** and **Explain Plan** tools, implies a big effort and a good knowledge of the application. Usually this step will improve the performance about 50%, fixing some of the "bottlenecks" on your system. After the tuning of the SQL statements have been done it is suggested, as a next step, to thoroughly analyze **Memory Allocation.**

Oracle stores information in two places:(a) on disk, (b) in memory. Since memory access is much faster than disk access, it is recommandable to access the memory instead of disk. The memory allocation analysis will be done on three different segments:

- Parsing and Context Areas,
- Data Dictionary Cache,

- Buffer Cache.

By proper sizing of parameters of these structures you can obtain improved performance.

PARSING & CONTEXT AREAS

An SQL statement is parsed before its execution. Oracle checks the statement for semantic and syntactic validity, checks the user privileges and will choose an execution plan.
Parsing is expensive, so it is recommended to be done as seldom as possible. During the statement parsing, the necessary information for the execution of the SQL statement is loaded into a **context area** which in fact represents workspace in memory. After an SQL statement is parsed, it can be executed repeatedly as long as the cursor information remains in the context area. To check the number of times an SQL statement is parsed use the **SQL Trace Facility** If the parsing number is high, it can be decreased by controlling the parsing frequency in your application. The memory allocation for **context area** is determined by the **init.ora** parameters:

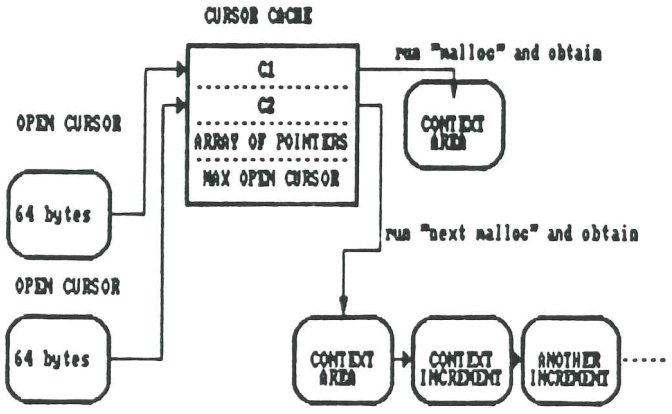
a. open_cursors
b. context_area
c. context_incr

Open_cursor: this parameter determines the total number of the context areas a single process can have, the default is 50.
Context_area: this parameter represents the initial size of context area.
Context_incr: this parameter represents the increment of space Oracle adds to a context area if the cursor information exceeds the **context_area**.

Oracle will dynamically increase the size of the context area until **context_incr** is large enough to hold the cursor information. The way the developer manipulates the cursors will have a direct impact to context areas. A cursor is a work area that Oracle uses for processing SQL statement(s). Pro*C uses two types of cursors:

a. Implicit cursors
b. Explicit cursors.

The way you handle the **opening** and **closing** of the explicit cursors affect performance. Usually, queries returning more than one row use explicit cursors by using one of the following commands: declare, open, fetch, and close. The flow can be summarized in the following way:



You can control very well the context areas and parsing by using different management strategies of the following parameters in the program:

a. hold_cursor

b. release_cursor

c. maxopencursors.

Hold_cursor specifies how the cursors for SQL statements and PL/SQL blocks are handled by the cursor_cache. When an SQL statement is executed, its associated cursor is linked to an entry in the cursor_cache, which in turn is linked to an Oracle context_area. Hold_cursor controls what happens to the link between the cursor and cache.

- IF **HOLD_CURSOR=NO**, then the cursor is closed and the precompiler marks the link as reusable. **This frees memory allocated to the context area and releases parse locks.**

  IF HOLD_CURSOR=YES, then the link is maintained. This is useful for **SQL statements that are executed often because it speeds up the executions.** There is no need to reparse the statement or allocate memory for a context area.
  Of course a **RELEASE_CURSOR=YES** will override **HOLD_CURSOR=YES.**
  Hold_cursor deals with the link between cursor and cache entry, and release_ cursor deals with the link between context area and cache entry.
  If a new cursor is needed and there are NO free cache entries, the precompiler tries to reuse an entry. The success depends how the parameters **hold_cursor** and **release_cursor** have been set. If it cannot reuse an old space, or the **MAXOPENCURSOR** limit is reached, the precompiler will temporarily allocate an additional cache entry. **This dynamic allocation adds to processing overhead.**

  - IF **MAXOPENCURSOR** is low, you can save memory but this causes dynamically expensive allocations and deallocations of new cache entries.
  - IF **MAXOPENCURSOR** is high, it assures speedy execution, but the user will need more memory.

The following rules are recommended:

- For *Frequently executed* SQL statements specify:
  HOLD_CURSOR = YES
  The link between the cursor and cursor cache is maintained, then the parsed statement and allocated memory remain available.
- For *Infrequently executed* SQL statements specify:
  HOLD_CURSOR = NO

The cursor is closed, the precompiler marks the link between the cursor and cache as reusable. The following table shows how these parameters indirect.

HOLD_CURSOR	RELEASE_CURSOR	LINKS ARE
NO	NO	marked as reusable
YES	NO	maintained
NO	YES	removed immediately
NO	YES	removed immediately

There are also two parameters (used in Oracle Call Interfaces) which have significant impact on context areas:

1. OOPEN
2. OSQL3

OOPEN opens the specified cursor. The syntax may also include the **area_size parameter.** This is used to specify a size for context area other than default.
OSQL3 is used to parse an SQL statement and associate it with a cursor. It contains the

address of a string containing the statement and the length of that statement. The way you define the **area_size parameter** can have an impact on the performance of the system.

DATA DICTIONARY CACHE

The area, located in the SGA(System Global Area), designated to hold dictionary data is known as *Data Dictionary Cache*. It comprises several caches, each holding information for a particular database object. The size of each data dictionary cache is determined by the **INIT.ora** parameter. Basically, when a user issues an SQL statement, that comes a data dictionary *MISS*, Oracle must execute additional SQL Statements to query dictionary tables. These calls are called "*recursive calls*" because they call themselves. While your application is running , there may be some dictionary misses. How would you check all these misses?
To check the current status of the dictionary cache, query the **V_$ROWCACHE** table. An SQL script can be run on this table and the results can be summarized in the following:

PARAMETER	GETS	MISSES	%MISSED	USED	COUNT	%USED
columns	2489984	802066	0.32	1998	3500	57
free_extents	2096	33	1.574	33	400	8
indexes	865420	164	0.018	184	500	37
tables	662441	138	0.020	159	400	40

The information in the **V_$ROWCACHE** is more complex. For our own purposes we will use the following parameters: parameter, gets, getmisses, count and usage.
PARAMETER: Represents the name of each cache entry.
GETS: Represents the number of requests for information.
GETMISSES: Represents the number of data requests, resulting in cache misses (the number of gets that involved a Disk Access.
COUNT: Represents the number of entities in a particular cache and is set in the init.ora.
After the program is run and a value for these parameters is obtained, it is recommended to use the following guidelines:

- Monitor the ratio (MISS_RATE) between the **GETMISSES** and **GET**. It is recommended that this percentage be less than 15%.
- Monitor the **COUNT** and **USAGE** columns. If the **USAGE** value is significantly less than the count value. then the cache hasn't been filled, so you can afford to reduce the number of cache entries.
- If the **MISS_RATE > 15%** and **USAGE = COUNT** then the parameter is a candidate for being increased. The reason is because all cache entries have been used and any new requests will go to disk.
- If the **MISS_RATE > 15%** and **USAGE < COUNT** the parameter is a candidate for being decreased. In this case a high **MISS_RATE** indicates the kernel is going to disk to fill up the cache, and the count for that parameter may be too high.

TUNING THE BUFFER CACHE

The **Buffer Cache** is an area in SGA that holds copies of the database blocks which contain data from: tables, indexes, rollbacks etc. The number of buffers in the cache is determine by the init.ora parameter **DB_BLOCK_BUFFERS**. Let's consider the following scenario: a user is retrieving data from a specific table. A block of data will be brought to the SGA database buffers. If a second user is looking for some data from the same table, which is already in the buffer, then the second user will get a better response time. Basically, the rule of thumb is that if the requests are satisfied by the blocks in the database buffer, the response time will be faster than reading blocks from disk. This implies that for a better

access you need to have enough buffers available. Unfortunately in a big application with many tables it is difficult to satisfy this request. Oracle collects some statistics to check this type of access. This can be done by checking the "Monitor I/O display", specifically checking the ratio between the logical and physical reads and writes as in the following example:

LOGICAL READS	PHYSICAL READS
7418580	137736

The **HIT_RATIO** = (Logical Reads - Physical Reads) / Logical Reads
Oracle recommends a constant checking of this parameter. If the **hit_ratio** is low, less than 70%, then it is necessary to **increase** the number of buffers in the Oracle cache to improve the performance. The only problem with increasing the **DB_BLOCK_BUFFERS** is that this will cause a **paging problem** in your system if there is not enough memory to hold a large buffer pool.
Oracle provides some "X$" tables to help the user check the actual performance of gain that more buffers will produce.

Guidelines and Conclusions on Buffer Cache

1. Run the **Monitor I/O** using the SQL*DBA MONITOR and check for the **hit_ratio**. If the hit_ratio is less than 70%, increase the parameter **DB_BLOCK_BUFFER.**
2. This increase will have a big impact on performance in your system *only* if the system has enough memory. By increasing this parameter the end user response time will increase, by reducing the number of I/O operations to the disk.
3. If your system is *short* on memory increasing the number of buffers will make the response time *worse* by increasing the paging I/O rate.

GENERAL CONCLUSIONS

Tuning the **Memory Allocation** is the second important step in tuning any Oracle database system. This paper describes three different approaches and it recommends that they be implemented, in the order they have been presented.

1. Start with the **Tuning Parsing and Context Area**. This will affect the frequency of access to the data dictionary. Identify the unnecessary parsing by reducing it with the Oracle Precompilers (hold_cursor, release_cursor, maxopencursor) and Oracle Call Interfaces (oopen and osql3).
2. Continue by tuning the data dictionary cache by examining the recursive calls and the cache misses. Special attention must be given to the **COUNT** and **USAGE** columns. By reducing the cache size, a lot of memory can be saved and can be reallocated to other memory structures.
3. End your memory tuning process by examining the **Buffer Cache**. Analyze the impact produced to your system by increasing the **DB_BLOCK_SIZE** parameter.

This process is a continuos process. Reallocate the available memory and tune back your operating system. Check the results, especially those related to swapping and paging.

In the production field there are times when the problem must be solved ad-hoc and a step by step methodology is hardly required. The intention of the author was to define such a methodology, although he is aware that there are not guaranteed techniques to improve the performance. This methodology was not developed by the author. It represents a selective

compilation of some of the best ideas to this problem in this field (see references). Based on his extensive experience with the software system design his only merit was to streamline the best approaches to this problem and to present it to the readers in a synthesized manner. The author used the above guidelines to tune a large application in a client/server environment.

REFERENCES

1. Tom Childers, Improving Access to Different Types od Tables, Oracle Magazine, vol 2, 1991
2. Michael Abbey, Tuning the DC Cache, Oracle User Resource, April 1991
3. Michael Potochnik, Techniques and Tools for Mastering your Oracle Applications, International Oracle Proceedings, 1990
4. Viorel Vlad, Tuning the SQL Statements in a Oracle client/server application, 17th ARA Congress, Los Angeles, June 1992
5. Oracle Corporation, Performance Tuning Guide, Oracle RDBMS Database Administrator's Guide, PRO*C etc.